Java IO、NIO、AIO

Blocking I/O、Non-blocking I/O、Asynchronous I/O

Jean Hu - 17 December 2016

前言

- Java 1.0 就提供 IO，支持輸入輸出。
- Java 1.4 後提供 NIO，支持非阻塞 IO (Non-blocking I/O)。
- Java 1.7 後提供 NIO2，支持非同步 IO (Asynchronous I/O)，需要作業系統支持。

目的

- 了解 Java 對非阻塞和非同步 IO 的支持。
- 了解不同 IO 適用的連線架構。

開始前準備

本架構建立於以下版本的環境：

- JDK8
- IntelliJ IDEA 2016.2

Blocking I/O

一. 模型

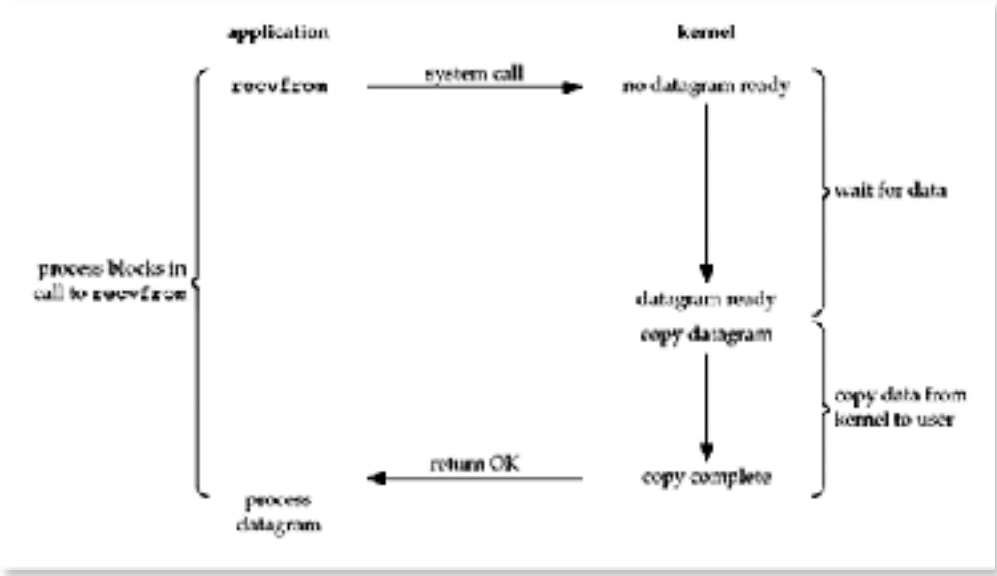

圖 1、Blocking I/O Model

- 舉個例子來說：
  1. Wait for data - 排隊等麥當勞點餐。
  2. Copy data from kernel to user - 點完餐後，等店員製作餐點。
- 適用的連線架構：連線數目較小且固定的架構。

二. Socket Client：三個例子都統一用 AioSocketClient 當 client 端。

```
AioSocketClient  main()
public class AioSocketClient {
    private static final int PORT = 7878;
    public static void main(String[] args) {
        try (AsynchronousSocketChannel channel = AsynchronousSocketChannel.open() ) {
            Future<Void> result = channel.connect(new InetSocketAddress("localhost", PORT));
            result.get();
            System.out.printf("[%s] Connect success!\n", Thread.currentThread().getName());
            ByteBuffer buff = ByteBuffer.allocate(256);
            Charset cs = Charset.forName("UTF-8");
            String msg = "Aloha copy cat";
            byte[] data = msg.getBytes(cs);
            buff.put(data);
            buff.flip();

            channel.write(buff, buff, new WriteHandler());
            Thread.currentThread().join();
        } catch (IOException | InterruptedException | ExecutionException ex) {
            ex.printStackTrace();
        }
    }
}
class WriteHandler implements CompletionHandler<Integer, ByteBuffer> {
    @Override
    public void completed(Integer result, ByteBuffer buff) {
        System.out.printf("[%s] Send message success!\n", Thread.currentThread().getName());
        buff.clear();
    }
    @Override
    public void failed(Throwable e, ByteBuffer buff) {
        e.printStackTrace();
    }
}
```

圖 2、AioSocketClient - Asynchronous I/O socket client

三. Socket Server

```
BioSocketServer
public class BioSocketServer {
    private static final int PORT = 7878;
    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            System.out.printf("[%s] Waiting connection....\n", Thread.currentThread().getName());
            Socket con = serverSocket.accept();  ——→  Block
            while(true) {
                System.out.printf("[%s] Waiting message....\n", Thread.currentThread().getName());
                BufferedInputStream in = new BufferedInputStream(new DataInputStream(con.getInputStream()));
                byte[] buff = new byte[256];
                int length;
                if ( ( length = in.read(buff) ) > 0) {  ——→  Block
                    System.out.printf("[%s] Read message from client: ", Thread.currentThread().getName());
                    System.out.println(new String(buff, 0, length, Charset.forName("UTF-8"))),
                }
            }
        } catch (IOException e) { e.printStackTrace(); }
    }
}
```

圖 3、BioSocketServer - Blocking I/O socket server

四. Run on Java Application：socket client 端統一用 AioSocketClient (參考圖 2)

```
Run:  BioSocketClient  AioSocketClient  BioSocketServer
"C:\Program Files\Java\jdk1.8.0_65\bin\java" ...
[main] Waiting connection....
[main] Waiting message....
[main] Read message from client: Aloha copy cat!
[main] Waiting message....
```

圖 4、Console log - BioSocketServer

Non-blocking I/O

一. 模型

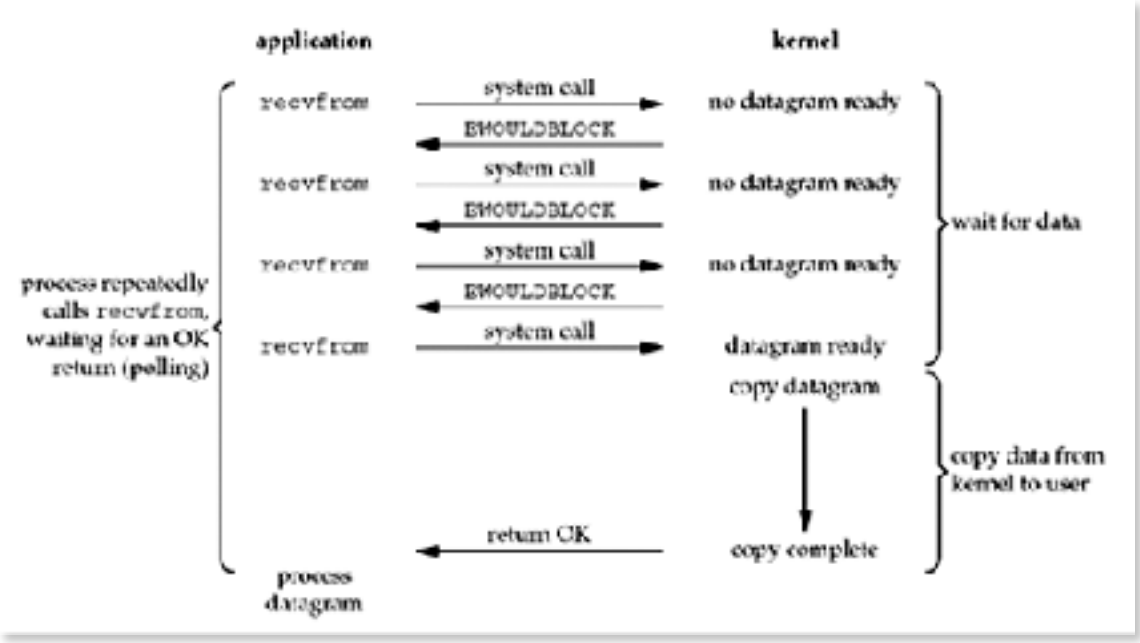

圖 5、Non-blocking I/O Model

- 舉個例子來說：
  1. Wait for data - 排隊邊等麥當勞點餐，邊講手機。
  2. Recvfrom system call - 時不時注意一下排到沒。
  3. Copy data from kernel to user - 點完餐後，等店員製作餐點。
- 適用的連線架構：連線數目較多且為短連線 (short-lived connection)。

二. Socket Server

```
NioSocketServer   main()
public class NioSocketServer {
    private static final int PORT = 7878;
    public static void main(String[] args) {
        try{
            ServerSocketChannel socketChannel = ServerSocketChannel.open();
            Selector selector = Selector.open()
        ) {
            socketChannel
                .bind(new InetSocketAddress("127.0.0.1", PORT))
                .configureBlocking(false) ——→ 設定Non-blocking
                .register(selector, socketChannel.validOps()), ——→ 註冊channel和事件
            while (true) {
                System.out.printf("[%s]Wait to be selected...\n", Thread.currentThread().getName());
                selector.select(); ——→ Block 直到事件發生
                Iterator<SelectionKey> selKeySet = selector
                        .selectedKeys()
                        .iterator();
                while (selKeySet.hasNext()) {
                    SelectionKey selKey = selKeySet.next();
```

```
if (selKey.isAcceptable()) {  ⟶ 新連線請求
    System.out.printf("[%s] Accept connection...\n", Thread.currentThread().getName());
    SocketChannel channel = socketChannel.accept();
    channel
            .configureBlocking(false)  ⟶ 設定Non-blocking
            .register(selector, SelectionKey.OP_READ);  ⟶ 註冊channel和讀取事件
} else if (selKey.isReadable()) {  ⟶ 讀取請求
    try (SocketChannel readChannel = (SocketChannel) selKey.channel()) {
        ByteBuffer buff = ByteBuffer.allocate(256);
        System.out.printf("[%s] Message from client: ", Thread.currentThread().getName());
        readChannel.read(buff);
        buff.flip();
        while (buff.hasRemaining()) {
            System.out.print((char) buff.get());
        }
        buff.clear();
        System.out.println();
    }
}
selKeySet.remove();
}
}
} catch (IOException ex) {
    ex.printStackTrace();
}
}
}
```

圖 6、NioSocketServer - Non-blocking I/O socket server

三. Run on Java Application：socket client 端統一用 AioSocketClient (參考圖 2)

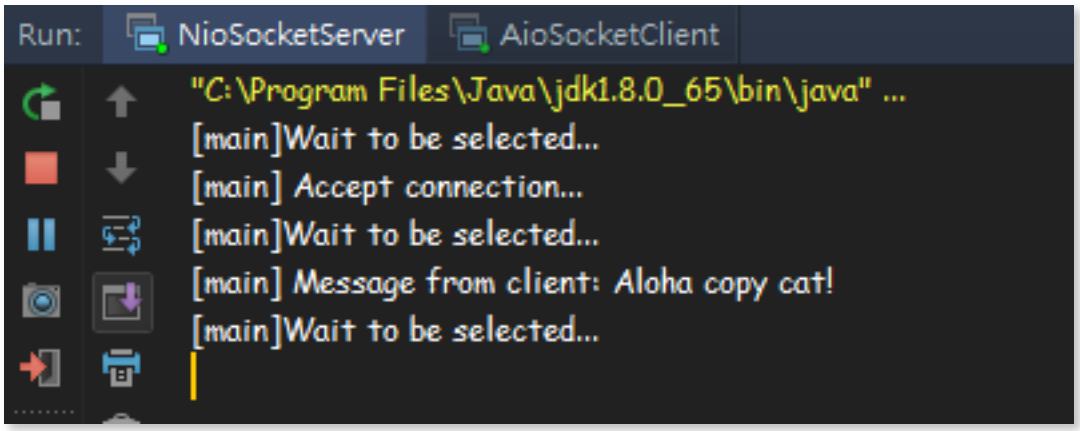

圖 7、Console log - NioSocketServer

Asynchronous I/O

一. 模型

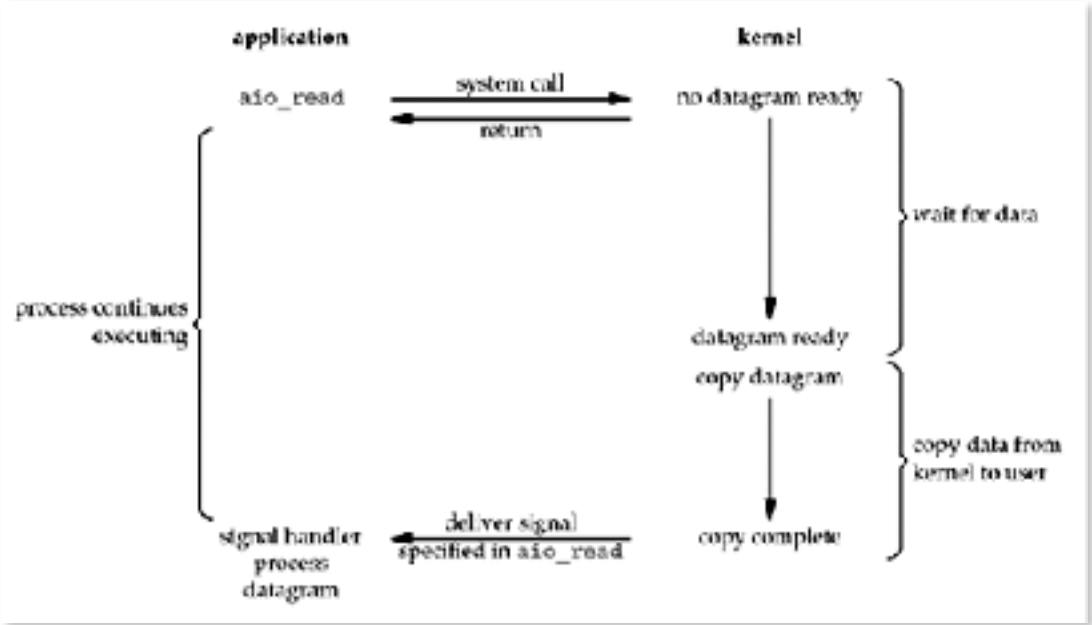

圖 8、Asynchronous I/O model

- 舉個例子來說：
  1. Aio_read system call - 吩咐倒茶小妹去幫忙買麥當勞，繼續做其他事。
  2. Wait for data- 倒茶小妹排隊等麥當勞點餐。
  3. Copy data from kernel to user- 倒茶小妹點完餐後，等店員製作餐點。
  4. Copy complete deliver signal- 倒茶小妹買完後通知。
- 適用的連線架構：連線數目較多且為長連線 (persistent connection)。

二. Socket Server

```
AioSocketServer
public class AioSocketServer {
    private static final int PORT = 7878;
    public static void main (String[] args) {
        AioSocketServer socketServer = new AioSocketServer();
        socketServer.runServer();
    }
    private void runServer() {
        try {
            AsynchronousServerSocketChannel server =
                AsynchronousServerSocketChannel.open().bind(new InetSocketAddress("127.0.0.1", PORT));
        } {
            server.accept(server, new ConnectionHandler());  ⟶ 採用Callback方式，異步接收client的連接
            while(true) {
                System.out.printf("[%s]Busy doing other things...\n", Thread.currentThread().getName());
                Thread.sleep(10000);
            }
//          Thread.currentThread().join();
        } catch (IOException | InterruptedException e) { e.printStackTrace(); }
    }
```

```
private class ConnectionHandler implements CompletionHandler<AsynchronousSocketChannel, AsynchronousServerSocketChannel> {
    @Override                                                                  ⟶ Handle異步操作，接收client端連接
    public void completed(AsynchronousSocketChannel socketChannel, AsynchronousServerSocketChannel serverSocketChannel) {
        System.out.printf("[%s] Accept connection success!\n", Thread.currentThread().getName());
                                                                               ⟶ 操作結果回傳值
                                                                               ⟶ Attachment
        serverSocketChannel.accept(serverSocketChannel, this);                 ⟶ 繼續接收下一個client端連接
        ByteBuffer buff = ByteBuffer.allocate(256);
        socketChannel.read(buff, buff, new ReadHandler());  ⟶ 使用Callback方式，異步讀取資料
    }

    @Override
    public void failed(Throwable e, AsynchronousServerSocketChannel serverSocketChannel) { e.printStackTrace(); }  ⟶ 連接失敗處理
}
```

```
private class ReadHandler implements CompletionHandler<Integer, ByteBuffer> {
    @Override
    public void completed(Integer result, ByteBuffer buff) {
        System.out.printf("[%s] Result: %d, Read message from client: ", Thread.currentThread().getName(), result);
        buff.flip();
        while (buff.hasRemaining()) {
            System.out.print((char) buff.get());
        }
        buff.clear();
        System.out.println();
    }

    @Override
    public void failed(Throwable e, ByteBuffer buff) { e.printStackTrace(); }
  }
}
```

圖 9、AioSocketServer - Asynchronous I/O socket server

三. Run on Java Application：socket client 端統一用 AioSocketClient (參考圖 2)

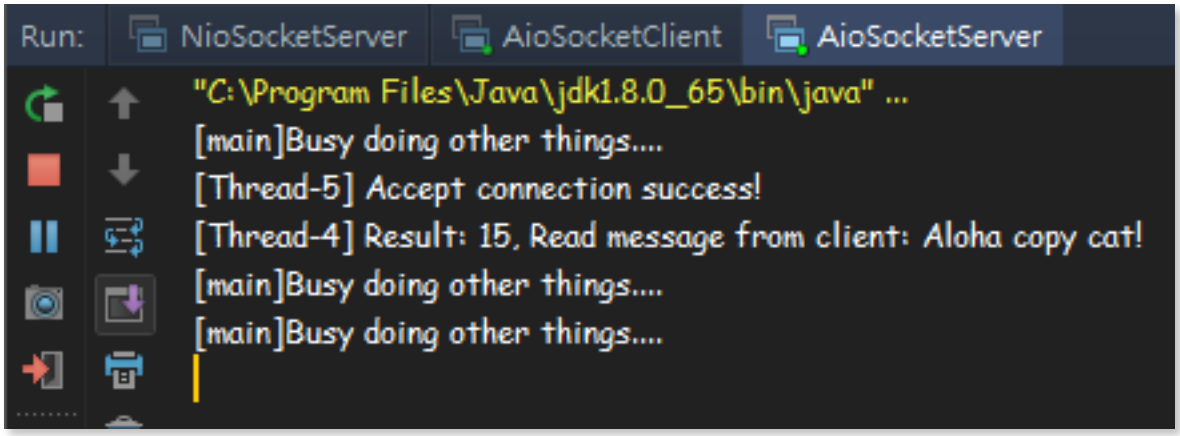

圖 10、Console log - AioSocketServer

參考來源

- I/O Models - https://notes.shich a o.io/unp/ch6/
- Java I/O, NIO, and NIO.2 - https://docs.oracle.com/javase/8/docs/technotes/guides/io/
- Java NIO package doc - https://docs.oracle.com/javase/8/docs/api/java/nio/package-summary.html
- Java NIO Tutorial - http://tutorials.jenkov.com/java-nio/index.html